02

Overview herencia

Transcripción

[00:00] Y bien. Si has llegado hasta este punto, si has visto todos los videos, pues felicidades, ya estás solamente a un paso de ya terminar este módulo, este curso, que es herencia y polimorfismo. Vamos a hacer un overview rápido de lo que hemos estado viendo hasta el momento. Hemos visto ya que las interfaces como ya vieron, son parecidas a las clases abstractas, pero ¿en qué se diferencian?

[00:38] Ya lo hemos visto incluso en el video anterior. Atributos no pueden tener, constructores tampoco, y los métodos no pueden tener implementación. La interfaz define solamente métodos abstractos, entonces si tú pones aquí un abstract, no hay ningún problema, es aceptado pero con una interfaz todos los métodos son abstractos, entonces no se pone, lo ignoramos.

[01:03] De igual forma aquí en la definición de la interfaz. Toda interfaz es abstracta y entonces se omite este modificador porque toda interfaz es abstracta y no se puede redundar tanto. Ahora, ya que hemos visto ya los dos pilares de la herencia, que en este caso es la reutilización de código y el polimorfismo, básicamente extends e implements, que son los dos pilares de la herencia en Java y sabemos las diferencias entre una clase abstracta, una interfaz.

[01:38] Vamos a poner algunos ejemplos prácticos ahí de cómo podemos mejorar este código, que si se dan cuenta, si bien nuestro modelo que vemos aquí ya está limpio, escalable, está bien hecho, aún tiene ciertas fallas y esas fallas, si bien no son graves, es muy bueno que las tengan mapeadas o las

tengan en el radar, porque les va a dar un mejor punto de vista para seguir trabajando esos proyectos reales, ya digamos de la escuela, la universidad o del trabajo mismo.

[02:21] Los conceptos que estamos aprendiendo aquí se aplican en el trabajo del día a día, así que por ejemplo yo veo en mi trabajo con temas de interfaces, herencias de código legado, proyectos que yo ya tomo ya hechos, ya ves esto digamos el pan de cada día, entonces va a seguir bastante. Vamos a agarrar nuestra clase cliente, porque si bien implementamos el método autenticable, no hemos implementado los métodos aún de iniciarSesion y de setClave.

[02:56] Entonces vamos pues a implementar esos métodos para ver cómo quedaría. Simulando pues en tono real, digamos, yo estoy haciendo este feature, voy a terminar de hacer este feature. Para setear la contraseña, ya habíamos quedado pues que listo, íbamos a tener un atributo clave, private String clave. Y en el setClave íbamos a ponerle que this.clave iba a ser igual a la clave que está viniendo como parámetro.

[03:36] Es exactamente la misma implementación que había en el anterior. Y aquí en implementación vamos a hacerlo dinámico, vamos a decirle por ejemplo if, y aquí vamos a preguntarle: If this.clave es igual a clave, entonces return true. Si no, retorna falso. Y ya tenemos pues nuestro método para setear la clave. Aquí yo defino qué clave voy a usar y aquí yo defino pues mi inicio de sesión.

[04:11] Y para el caso del contador no hay nada porque él no inicia sesión, entonces lo dejo fuera, vamos con el administrador, porque el administrador también inicia sesión y también necesita una clave, aparte de que también se define su bonificación. Entonces, vamos a decirle que el administrador va a recibir bonificación como salario. getSalario, perfecto.

[04:37] Y similar al anterior, vamos a definir private String clave. Aquí, en el setClave, ¿qué vamos a hacer? This.clave, se me escapó una coma, punto clave

igual clave. ¿Sienten que últimamente empieza a oler un poco mal? ¿No? Aquí, nuevamente tenemos un problema. If. Y aquí le hacemos if this.clave es igual a clave, return true.

[05:19] Incluso si nosotros queremos hacerlo en una sola línea, dado que esa es una expresión booleana, podemos decirle simplemente return this.clave igual igual a la clave. Y esto lo borramos. También es válido. Pero básicamente estos códigos de acá, tanto este de aquí como este de aquí, son exactamente los mismos. Entre aquí y aquí.

[05:59] Son exactamente el mismo código. Ambos tienen su atributo clave y ambos setean clave de la misma forma y prácticamente ejecutan el mismo código, la misma lógica. Vamos a dejarlo igual. Y si bien digamos que no es un problema grave por ahora porque la cantidad de código pues es ínfima, es una línea, de cierta forma se está repitiendo y este es un antipatrón que no queremos seguir.

[06:31] Sabemos pues que el código lo queremos reutilizar lo más posible y no es bueno repetir el código entre métodos, entonces para solucionar esto, vamos a implementar una nueva clase que va a ser una clase útil. ¿Clase útil a qué nos referimos? Por clase útil estamos hablando de utilitarios, digamos, métodos que no le pertenecen a ninguna clase pero que tienen cierta funcionalidad pues para ayudar a ahorrar código y no repetir el código.

[07:08] Por ejemplo un utilitario puede ser obtener fecha actual. Si tú necesitas obtener la fecha actual en múltiples partes de tu sistema, entonces tú no amarras el método de obtener fecha actual a un objeto. Lo amarras a una clase útil. ¿Por qué? Porque el útil va a estar disponible para todos, es como un utilitario, es como una caja de herramientas.

[07:28] Entonces, al ser una caja de herramientas común, cada uno agarra la herramienta que necesite. Entonces definimos un AutenticacionUtil. Perfecto. Le damos a finish, y en autenticación útil vamos a definirle pues su clave.

Private String clave y AutenticaciónUtil va a tener su método público iniciarSesion que ¿qué va a recibir? Va a recibir un string clave. Listo.

[08:13] ¿Y él qué va a retornar? Prácticamente va a retornar lo mismo que estamos retornando acá, entonces le damos a copiar. ¿A qué quiero llegar con esto? Él va a retornar lo mismo de aquí y él va a tener public setClave. Aquí ya va String Clave, listo. Y él va a tener el mismo comportamiento aquí que this.clave va a ser igual a clave. Listo.

[08:50] Y acá es un void, error mío. Listo. ¿Entonces cuál es el fin de usar AutenticacionUtil? Ahorrar código. ¿Por qué? Porque acá en cliente yo podría ahora llamar esa clase AutenticacionUtil también.